PHYS 410 Project 2

Name: Steven Brown
Student Number: 90169772

1 Problem 1 Introduction

In problem one of this project the goal was to solve the one-dimensional time-dependent Schrodinger equation using the Crank-Nicolson discretization method. The Schrodinger equation is a linear partial differential equation that governs the wave function of a quantum-mechanical system. This wave function which we will call ψ is a function that assigns a complex number to each point x at each time t. The parameter $V(x,t)$ is the potential that represents the environment in which the particle exists. The full one-dimensional time-dependent equation can be seen below after non-dimensionalization.

$$i\psi(x,t)_t = -\psi_{xx} + V(x,t)\psi \tag{1}$$

This equation is to be solved on the domain

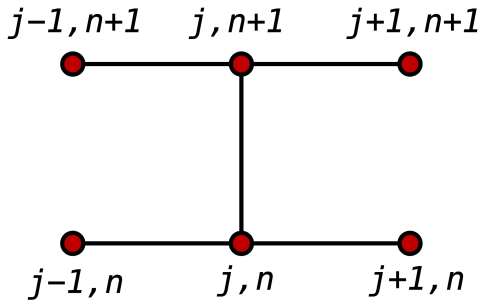
$$0 \leq x \leq 1, \quad 0 \leq t \leq t_{max}$$

subject to initial and boundary conditions

$$\psi(x,0) = \psi_0(x), \quad \psi(0,t) = \psi(1,t) = 0$$

It is important to note that the initial data function $\psi_0(x)$ can be complex in general, but in the cases we will consider, it will be real. In the actual solution will also restrict attention to time-independent potentials, but carry the explicit dependence in the following derivations to emphasize that it is no more difficult to deal with a time-dependent potential than a time-independent one.

The Crank–Nicolson method is a finite difference method used for numerically solving partial differential equations and is second-order in time. This method is based on the trapezoidal rule which gives second-order convergence in time. We can also note that for linear equations, the trapezoidal rule is equivalent to the implicit midpoint method. The following image is provided to help imagine the discretization scheme.



Note that code snippets will be included in this report but to see the full code and function descriptions see the .m files submitted.

2 Review of Theory & Numerical Approach for Problem 1

2.1 The One-Dimensional Time-Dependent Schrodinger Equation

As stated in the introduction, in this section we will be discussing the one-dimensional time-dependent Schrodinger equation and some of its useful properties and quantities. For reference, the equation under analysis is stated below.

$$i\psi(x,t)_t = -\psi_{xx} + V(x,t)\psi$$

2.1.1 Useful Quantities

The one-dimensional time-dependent Schrodinger equation has some useful properties and useful quantities that we will be discussing. We note that ψ can be expressed in terms of its real and imaginary parts, ψ_{Re} and ψ_{Im}, respectively as

$$\psi = \psi_{Re} + \psi_{Im}$$

A useful diagnostic quantity is the "running integral", $P(x,t)$, of the probability density, $\rho = |\psi|^2 = \psi\psi^*$, where * denotes the complex conjugate operator. Note that the ρ represents the probability density for the position of a particle in one dimension.

$$P(x,t) = \int_0^x \psi(\tilde{x},t)\psi^*(\tilde{x},t)d\tilde{x} \tag{2}$$

Note that if the wave function is properly normalized, then we will have

$$P(1,t) = 1$$

because this represents the probability that the particle is found in our entire domain at any point in time, which is 100%. Even if our quantity is not so normalized (and in our applications there will be no need to ensure normalization), we should have

$$P(1,t) = \text{conserved to level of solution error}$$

The quantity $\sqrt{\psi} = |\psi|$ is also a useful diagnostic. Additionally, it was found useful to plot the real and imaginary parts of ψ as well.

A family of exact solutions of (1) can be described by

$$\psi(x,t) = e^{-im^2\pi^2 t}\sin(m\pi x) \tag{3}$$

where m is a positive integer representing the particles mass.

The integral in (2) can be computed to $O(h^2)$ using the trapezoidal formula. Recall that if we are given n approximate values f_i at values of x, x_i, then the trapezoidal approximation is given by

$$\int_{x_1}^{x_n} f(x)dx \approx \frac{1}{2}\sum_{i=1}^{n-1}(f_i + f_{i+1})(x_{i+1} - x_i) \tag{4}$$

2.1.2 Useful Quantities Implementation

The useful quantities that were previously discussed were implemented with the following code inside the sch_1d_cnm̀ script. Note that the indexing was shifted when calculating the probability density matrix in order to calculate the last value using trapezoidal approximation. This keeps the first value as zero which makes sense for a "running integral".

```
psire = real(psi);
psiim = imag(psi);
psimod = sqrt(psi .* conj(psi));
prob = zeros(nt, nx);

% calculate running integral of the probability density
ro = psimod.^2;
for ii = 1 : nt
    sum = 0;
    for jj = 2 : nx
        sum = sum + 0.5 * (ro(ii, jj-1) + ro(ii,jj)) * (x(jj) - x(jj-1));
        prob(ii, jj) = sum;
    end
end
```

2.1.3 Initial Conditions and Potentials

Partial differential equations can be very sensitive to initial conditions. These initial conditions create solution families in different forms of the wave equation. Our potential $V(x,t)$ also effects the behaviour of the solution. For this problem we will be considering two initial data types and two potential functions. The two initial data types can be described as exact family and boosted Gaussian respectively.

$$\psi(x,0) = \sin(m\pi x) \tag{5}$$

$$\psi(x,0) = e^{ipx} e^{-((x-x_0)/\delta)^2} \tag{6}$$

The two potential functions can be described as no potential and rectangular barrier/well.

$$V(x) = 0 \tag{7}$$

$$V(x) = \begin{cases} 0 & \text{for } x \le x_{min} \\ V_c & \text{for } x_{min} \le x \le x_{max} \\ 0 & \text{for } x \ge x_{max} \end{cases} \tag{8}$$

The terminology "boosted Gaussian" comes from the fact that by pre-multiplying the (real-valued) Gaussian profile by e^{ipx}, we give the wave packet some momentum in the direction of p (which can have either sign). The constant V_c is positive for a barrier, negative for a well. In this problem, we will not worry about the fact that the boosted Gaussian profile is incompatible with the boundary conditions. In practice we center the Gaussian sufficiently far from the boundaries such that the incompatibility is lost in the truncation error. We always set the wave function to 0 at x = 0 and x = 1, including at the initial time.

2.1.4 Initial Conditions and Potentials Implementation

The input parameters idtype and vtype are integers that select which initial data type and potential type, respectively, are to be used. Dependent on the type, elements of the associated parameter vector will be used to define the initial data or potential. Specifically, implementation options as follows:

Initial Data Types

1. Exact Family (5): idtype == 0

- idpar(1) = m

2. Boosted Gaussian (6): idtype == 1

- idpar(1) = x_0
- idpar(2) = δ
- idpar(3) = p

Potential Types

1. No Potential (7): vtype == 0
2. Rectangular Barrier or Well (8): vtype == 1

- idpar(1) = x_{min}
- idpar(2) = x_{max}
- idpar(3) = V_c

The initial conditions and potentials that were previously discussed were implemented with the following code inside the sch_1d_cnm̀ script.

```
% Initial Data
if idtype == 0
    psi(1, :) = sin(idpar(1) * pi * x);
elseif idtype == 1
    psi(1, :) = exp(1i*idpar(3)*x) .* exp(-((x-idpar(1)) / idpar(2)).^2);
else
    fprintf('Invalid idtype');
    return
end

% Potential Data
if vtype == 0
    v = zeros(nx, 1);
elseif vtype == 1
    v = zeros(nx, 1);
    for idx = 1:nx
        if vpar(1) <= x(idx) && x(idx) <= vpar(2)
            v(idx) = vpar(3);
        end
    end
else
    fprintf('Invalid idtype');
    return
end
```

2.2 Crank-Nicolson Discretization

The Crank–Nicolson method is a finite difference method used for numerically solving partial differential equations using discretization. We discretize the continuum domain by introducing the democratization

level, l, and the ratio of temporal to spatial mesh spacings

$$\lambda = \frac{\Delta t}{\Delta x}$$

We then define the following parameters

$$n_x = 2^l + 1$$
$$\Delta x = 2^{-l}$$
$$\Delta t = \lambda \Delta x$$
$$n_t = \text{round}(t_{max}/\Delta t) + 1$$

We can apply the Crank-Nicolson discretization approach to (1) and its respective boundary conditions. We then define the FDA as the combination of a first order explicit scheme and a first order implicit scheme plugged into (1). This results in

$$i\frac{\psi_j^{n+1} - \psi_j^n}{\Delta t} = -\frac{1}{2}\left(\frac{\psi_{j+1}^{n+1} - 2\psi_j^{n+1} + \psi_{j-1}^{n+1}}{\Delta x^2} + \frac{\psi_{j+1}^n - 2\psi_j^n + \psi_{j-1}^n}{\Delta x^2}\right) + \frac{1}{2}V_j^{n+1/2}(\psi_j^{n+1} + \psi_j^n) \tag{9}$$

where

$$j = 2, 3, ..., n_x - 1, \quad n = 1, 2, ..., n_t - 1$$

We also defined the boundary conditions as

$$\psi_1^{n+1} = \psi_{n_x}^{n+1} = 0, \quad n = 1, 2, ..., n_t - 1 \tag{10}$$
$$\psi_j^1 = \psi_0(x_j), \quad j = 1, 2, ..., n_x \tag{11}$$

The solution to this equation can be found using a tri-diagonal system. We first rewrite the above equation in the form

$$c_j^+ \psi_{j+1}^{n+1} + c_j^0 \psi_j^{n+1} + c_j^- \psi_{j-1}^{n+1} = f_j$$

where the different c vectors are the values along the three central diagonals in the sparse matrix. After rearranging (9) we find

$$c_j^+ = \frac{1}{2\Delta x^2}$$
$$c_j^0 = \frac{i}{\Delta t} - \frac{1}{\Delta x^2} - \frac{V_j^{n+1/2}}{2}$$
$$c_j^- = \frac{1}{2\Delta x^2} = c_j^+$$
$$f_j = \frac{i}{\Delta t}\psi_j^n - \frac{1}{2}\left(\frac{\psi_{j+1}^n - 2\psi_j^n + \psi_{j-1}^n}{\Delta x^2}\right) + \frac{1}{2}V_j^{n+1/2}\psi_j^n$$

We can use these coefficient vectors to generate a sparse matrix using "spdiags" and then solve the system using left division.

2.3 Crank-Nicolson Discretization Implementation

The steps for the Crank-Nicolson discretization method previously discussed was implemented with the following code inside the sch_1d_cnm̀ script.

```
% Initialize storage for sparse matrix and RHS
dl = zeros(nx,1);
d  = zeros(nx,1);
du = zeros(nx,1);
f  = zeros(nx,1);

% Set up tridiagonal system
dl = 0.5 / dx^2 .* ones(nx, 1);
d  = (1i / dt - 1.0 / dx^2 - v / 2) .* ones(nx,1);
du = dl;

% Fix up boundary cases
d(1) = 1.0;
du(2) = 0.0;
dl(nx-1) = 0.0;
d(nx) = 1.0;

% Define sparse matrix
A = spdiags([dl d du], -1:1, nx, nx);

% Compute solution using CN scheme
for n = 1 : nt-1
    % Define RHS of linear system
    f(2:nx-1) = 1i * psi(n, 2:nx-1) / dt - 0.5 * ( ...
        psi(n, 1:nx-2) - 2 * psi(n, 2:nx-1) + psi(n, 3:nx)) / dx^2 ...
        + 0.5 * v(2:nx-1).' .* psi(n, 2:nx-1);
    f(1) = 0.0;
    f(nx) = 0.0;
    % Solve system, thus updating approximation to next time step
    psi(n+1, :) = A \ f;
end
```

2.4 Convergence Testing

Define a level l solution computed using sch_1d_cn by ψ^l. Note that ψ^l is a function of both the discrete time and space coordinates. Denote by $d\psi^l$ the quantity defined by

$$d\psi^l = \psi^{l+1} - \psi^l$$

where the data defined by the grid function (array) ψ^{l+1} is 2:1 coarsened in both the time and space dimensions so that it has the same size as ψ^l. Then one way we can convergence test is to compute

$$\|d\psi^l\|_2(t^n) \tag{12}$$

where $\|.\|_2$ denotes the l-2 norm (RMS value) that can be defined as the following for any length-m vector v.

$$\|v\|_2 = \sqrt{\frac{\sum_{j=1}^{m} |v^j|^2}{m}}$$

Observe that for complex numbers, $|v^j|$ is the modulus of the number. Note this makes sense because (12) involves taking spatial norms of the pairwise subtraction of grid functions at two different levels. This results in a function of the discrete time, t^n, on the level-l grid.

Following the development we have seen for solutions of other finite difference equations, we note that since our FDA is $O(h^2)$ (where $\Delta x = h$ and $\Delta t = \lambda h$), we expect the solution to be $O(h^2)$ accurate, with ψ^l following an expansion of the form

$$\psi^l(x,t) = \psi(x,t) + h_l^2 e_2(x,t) + O(h_l^4) \tag{13}$$

where $e_2(x,t)$ is some error function. From (13) we can deduce that if we graph re-scaled values of $\|d\psi^l\|_2$ on a single plot, then convergence is signalled by near-coincidence of the curves, with better agreement as we go to higher values of l. In particular, for a test with levels

$$l = l_{min}, l_{min}+1, ..., l_{max}$$

we then plot

$$\|d\psi^{l_{min}}\|_2, 4\|d\psi^{l_{min}+1}\|_2, 4^2\|d\psi^{l_{min}+2}\|_2, ..., 4^{l_{max}-l_{min-1}}\|d\psi^{l_{max}-1}\|_2$$

Additionally, for the case when idtype = 0, so that we know the exact solution, we can compute the actual solution errors. Specifically, we can perform precisely the same type of convergence test just described, but where ψ^{l+1} is replaced with ψ_{exact}. Thus we define

$$\|E(\psi^l)\|_2(t^n) = \|\psi_{exact} - \psi^l\|_2(t^n)$$

and use exactly the same plotting strategy for $\|E(\psi^l)\|_2$ as we do for $\|d\psi^l\|_2$.

2.5 Convergence Testing Implementation

The convergence test we preformed had the following parameters

1. idtype = 0, vtype = 0
   - idpar = [3]
   - tmax = 0.25
   - lambda = 0.1
   - $l_{min} = 6$
   - $l_{max} = 9$
2. idtype = 1, vtype = 0
   - idpar = [0.50 0.075 0.0]
   - tmax = 0.01
   - lambda = 0.01

- $l_{min} = 6$
- $l_{max} = 9$

The following is the function that will calculate the convergence test and plot the three graphs. The code was derived from the previous section.

```
function ctest_1d()
    % get solutions for convtest 1 for different levels
    m = 3;
    [x1_6 t1_6 psi1_6 psire1_6 psiim1_6 psimod1_6 prob1_6 v1_6] = ...
        sch_1d_cn(0.25, 6, 0.1, 0, [m], 0, []);
    [x1_7 t1_7 psi1_7 psire1_7 psiim1_7 psimod1_7 prob1_7 v1_7] = ...
        sch_1d_cn(0.25, 7, 0.1, 0, [m], 0, []);
    [x1_8 t1_8 psi1_8 psire1_8 psiim1_8 psimod1_8 prob1_8 v1_8] = ...
        sch_1d_cn(0.25, 8, 0.1, 0, [m], 0, []);
    [x1_9 t1_9 psi1_9 psire1_9 psiim1_9 psimod1_9 prob1_9 v1_9] = ...
        sch_1d_cn(0.25, 9, 0.1, 0, [m], 0, []);

    % coarsen solutions to match size of lmin solution
    psi1_7 = psi1_7(1:2:end, 1:2:end);
    psi1_8 = psi1_8(1:4:end, 1:4:end);
    psi1_9 = psi1_9(1:8:end, 1:8:end);

    dpsi6 = psi1_7 - psi1_6;
    dpsi7 = psi1_8 - psi1_7;
    dpsi8 = psi1_9 - psi1_8;

    % calculate rms values of dpsi
    rms_dpsi6 = rms(dpsi6, 2);
    rms_dpsi7 = rms(dpsi7, 2);
    rms_dpsi8 = rms(dpsi8, 2);

    % Removed all non-plot lines for clarity
    plot(t1_6, rms_dpsi6, 'r-.o');
    plot(t1_6, 4 * rms_dpsi7, 'g-.+');
    plot(t1_6, 4^2 * rms_dpsi8, 'b-.*');

    % calculate exact solution
    psi_exact = zeros(size(t1_6, 2), size(x1_6, 2));
    for idx = 1:size(t1_6, 2)
        psi_exact(idx, :) = exp(-1i * m^2 * pi^2 * t1_6(idx)) * sin(m * pi * x1_6);
    end

    % calculate rms values of E
    rms_E6 = rms(psi_exact - psi1_6, 2);
    rms_E7 = rms(psi_exact - psi1_7, 2);
    rms_E8 = rms(psi_exact - psi1_8, 2);
    rms_E9 = rms(psi_exact - psi1_9, 2);

    % Removed all non-plot lines for clarity
    plot(t1_6, rms_E6, 'r-.o');
    plot(t1_6, 4 * rms_E7, 'g-.+');
```

```
plot(t1_6, 4^2 * rms_E8, 'b-.*');
plot(t1_6, 4^3 * rms_E9, 'b-.*');
```

end

2.6 Numerical Experiments

To make sense of our results we ran through some numerical experiments. The values calculated in these experiments are as follows. Consider the discrete running integral of the probability density

$$P_j^n = P(x_j, t^n), \quad j = 1, 2, ..., n_x, \quad n = 1, 2, ..., n_t)$$

We define the temporal average, $\overline{P}_j$ of the above quantity:

$$\overline{P}_j = \frac{\sum_{n=1}^{n_t} P_j^n}{n_t}$$

We will also want to ensure that $\overline{P}_j$ is properly normalized so that $\overline{P}_{n_x} = 1$. We can do this as follows:

$$\overline{P}_j := \frac{\overline{P}_j}{\overline{P}_{n_x}}, \quad j = 1, 2, ..., n_x$$

In the following we will assume that $\overline{P}_j$ has been properly normalized. Given two values of x, x_1 and x_2, satisfying $x_2 \geq x_1$, we can interpret the quantity

$$\overline{P}(x_2) - \overline{P}(x_1)$$

as the fraction of time our quantum particle spends in the interval $x_1 \leq x \leq x_2$. Here the notation $\overline{P}(x)$ is to be interpreted as $\overline{P}(x) = \overline{P}(x_j) = \overline{P}_j$ where x_j is the nearest grid point to x. Now, for a free particle—i.e. for $V = 0$ and for sufficiently long times, we expect that

$$\overline{P}(x_2) - \overline{P}(x_1) \to x_2 - x_1$$

That is, the fraction of time the particle spends in the interval is given simply by the width of the interval (this direct equality is due to the fact that we are solving the Schrodinger equation on the unit interval, $0 \leq x \leq 1$). For the general case of a non-zero potential, we can then define the excess fractional probability that the particle spends in a given spatial interval as

$$\overline{F}_e(x_1, x_2) = \frac{\overline{P}(x_2) - \overline{P}(x_1)}{x_2 - x_1}$$

For the experiments described in the next section, this quantity will span orders of magnitude so for the purposes of plotting it will be convenient to compute its natural logarithm

$$\ln \overline{F}_e(x_1, x_2) = \ln \frac{\overline{P}(x_2) - \overline{P}(x_1)}{x_2 - x_1}$$

We can also observe that although we have called $\overline{F}_e(x_1, x_2)$ the excess fractional probability, it can in fact satisfy $\overline{F}_e(x_1, x_2) \leq 1$, indicating that the particle is spending less time in the specified interval than a free

particle would. Indeed, in the experiments that follow, we found find that $\overline{F}_e(x_1, x_2) \leq 1$ is the rule rather than the exception.

2.7 Numerical Experiments Implementation

The numerical experiments preformed had the following parameters

1. Barrier Survey

- tmax = 0.1
- level = 9
- lambda = 0.01
- idtype = 1
- idpar = [0.40, 0.075, 20.0]
- vtype = 1
- vpar = [0.6, 0.8, VARIABLE ≥ 0]
- $x_1 = 0.8$
- $x_2 = 1.0$

2. Well Survey

- tmax = 0.1
- level = 9
- lambda = 0.01
- idtype = 1
- idpar = [0.40, 0.075, 0.0]
- vtype = 1
- vpar = [0.6, 0.8, VARIABLE ≤ 0]
- $x_1 = 0.6$
- $x_2 = 0.8$

The following is the function that will calculate the numerical experiments and plot the specified graphs. The code was derived from the previous section.

```
% calculate output vectors
Fe = zeros(num_exp, 1);

% run experiments
for idx = 1:num_exp
    [x t psi psire psiim psimod prob v] = ...
        sch_1d_cn(tmax, level, lambda, 1, [0.40, 0.075, 20.0], 1, [0.6 0.8 V0(idx)]);

    % p_j^n = prob[x_j, t^n] is the convention
    % calculate normalized temporal average
    temp_avg_prob = mean(prob);
    temp_avg_prob = temp_avg_prob / temp_avg_prob(nx);
```

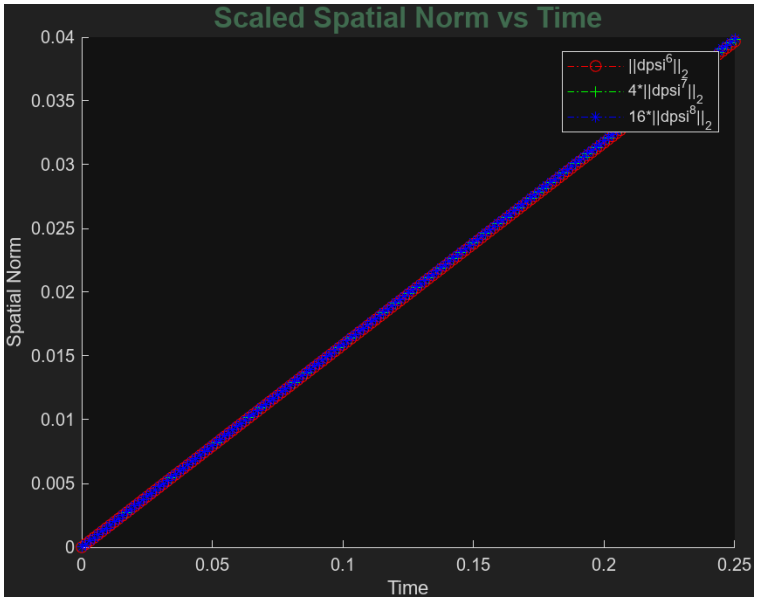
```
        % find closest points in x to x1 and x2
        [~, x1_idx] = min(abs(x - x1));
        [~, x2_idx] = min(abs(x - x2));

        % calculate fractional probability
        Fe(idx) = (temp_avg_prob(x2_idx) - temp_avg_prob(x1_idx)) / (x2 - x1);
        if Fe >= 1
            fprintf("Broken Rule");
        end

        if mod(idx, 30) == 0
            fprintf("Iteration " + string(idx) + "\n");
        end
    end

    % Removed all non-plot lines for clarity
    plot(lnV0, log(Fe), 'r-.o');
```

This code was used in both barrier_survey.m and well_survey.m to generate their respective plots with their respective initial conditions.

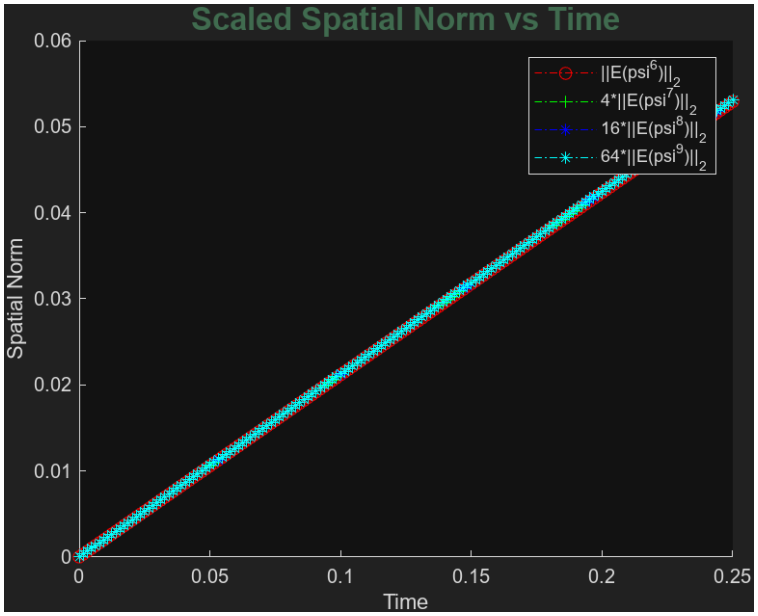
3 Problem 1 Results

3.1 Convergence Test

The result of the 4-level convergence test can be seen blow. Note that we do indeed see near-coincidence of the curves, with better agreement as we go to higher values of l. As a stringent test that our numerical solution is behaving as it should, implying that the implementation is correct, we performed a 4-level (6, 7, 8 and 9) convergence test for our FDA as discussed in class following the steps discussed in the previous section. With the CrankNicolson algorithm developed, a 4-level convergence test was performed and we noticed near-coincidence of the curves, with better agreement as we go to higher values of l. This was done again but comparing the exact value of the solution and again near-coincidence of the curves was observed.



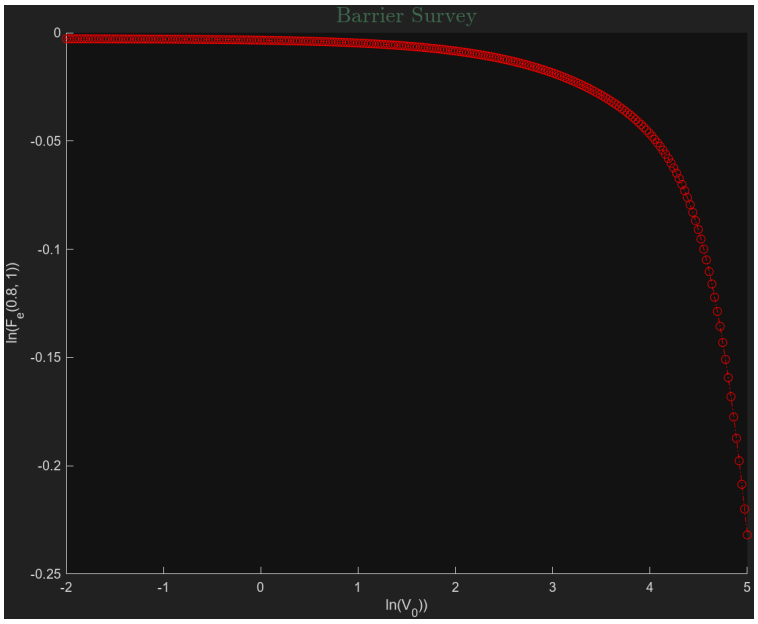
The following graph is for the case of using ψ_{exact} in our 4-level convergence test calculation.



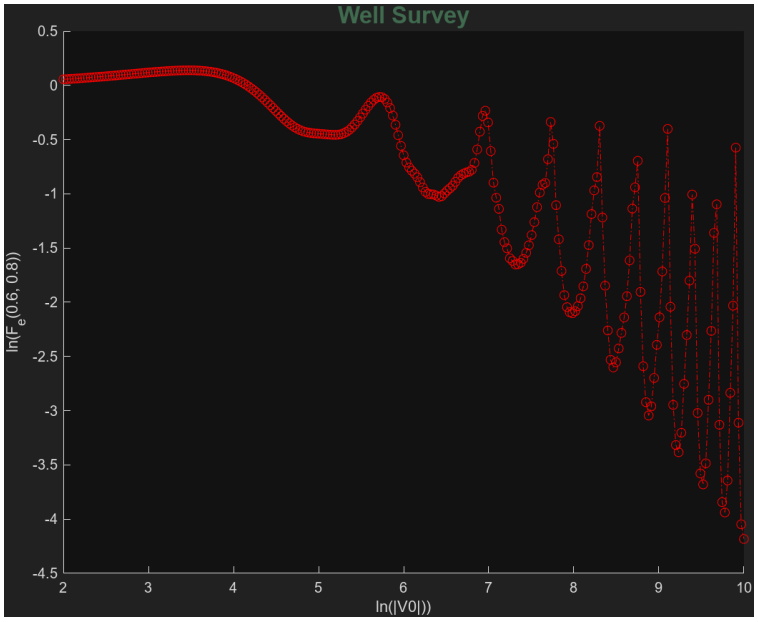
If we wanted to complete a more detailed convergence test we could extend our convergence test by repeating the calculation with an even higher levels.

3.2 Numerical Experiments

The result from the barrier and well survey experiments can be seen below. For the barrier survey case, we computed $\overline{F}_e(0.8, 1.0)$ for 251 uniformly spaced values of $\ln(V_0)$ ranging from -2 to 5. We went on to perform two numerical experiments where the excess fractional probability that the particle spends in a given spatial interval was analyzed. For the experiments described, this quantity will span orders of magnitude so, particularly for the purposes of plotting, it was convenient to compute its (natural) logarithm. These values were graphed for the two experiments.



For the well survey case, we computed $\overline{F}_e(0.6, 0.8)$ for 251 uniformly spaced values of $\ln(|V_0|)$ ranging from 2 to 10.



If we wanted to complete a more detailed experiment we could extend our experiment by repeating the calculation with an even higher levels and less spacing between the values of V_0. It made sense that for the barrier survey, the graph started a bit below zero and then dropped off with increasing values of V0. For the well survey, it made sense that it started a bit above zero and then exhibited some sinusoidal tendencies while also dropping in value as we increased V0.

4 Problem 2 Introduction

In problem two of this project our goal was to solve the two-dimensional Schrodinger equation using the alternating direction implicit (ADI) discretization technique. The non-dimensionalized continuum equation we will be analyzing is

$$\psi_t = i(\psi_{xx} + \psi_{yy}) - iV(x,y)\psi \tag{14}$$

In this form, the similarity to a diffusion equation with an imaginary diffusion constant (and a source term) can be seen. Equation (14) is to be solved on the domain

$$0 \leq x \leq 1, \quad 0 \leq y \leq 1, \quad 0 \leq t \leq t_{max}$$

subject to initial and boundary conditions

$$\psi(x,y,0) = \psi_0(x,y)$$
$$\psi(0,y,t) = \psi(1,y,t) = \psi(x,0,t) = \psi(x,1,t) = 0$$

5 Review of Theory & Numerical Approach for Problem 2

5.1 The Two-Dimensional Schrodinger Equation

As stated in the introduction, in this section we will be discussing the two-dimensional Schrodinger equation and some of its useful properties and quantities. Note that most of the useful properties are the same as for the one dimensional case.

5.1.1 Useful Quantities

The most important different useful quantity from the one-dimensional case is a different family of exact solutions for the wave equation. Most of the other quantities are the same idea with different dimensions. A family of exact solutions of (14) with its respective initial conditions and boundary conditions can be described by

$$\psi(x,y,t) = e^{-i(m_x^2+m_y^2)\pi^2 t}\sin(m_x\pi x)\sin(m_y\pi y) \tag{15}$$

where m_x and m_y are positive integers.

5.1.2 Useful Quantities Implementation

The useful quantities that were previously discussed were implemented with the following code inside the sch_2d_adi.m script.

```
psire = real(psi);
psiim = imag(psi);
psimod = sqrt(psi .* conj(psi));
```

5.2 Initial Conditions and Potentials

Partial differential equations can be very sensitive to initial conditions. These initial conditions create solution families in different forms of the wave equation. Our potential $V(x, y, t)$ also effects the behaviour of the solution. For this problem we will be considering two initial data types and three potential functions. The two initial data types can be described as exact family and boosted Gaussian respectively.

$$\psi(x, y, 0) = \sin(m_x \pi x) \sin(m_y \pi y) \tag{16}$$

$$\psi(x, y, 0) = e^{ip_x x} e^{ip_y y} e^{-((x-x_0)^2/\delta_x^2 + (y-y_0)^2/\delta_y^2)} \tag{17}$$

As previously, we will not worry about the fact that the Gaussian data is strictly speaking incompatible with the boundary conditions; just be sure to always impose the correct boundary conditions. The three potential functions can be described as no potential, rectangular barrier/well, or double slit.

$$V(x, y) = 0 \tag{18}$$

$$V(x, y) = \begin{cases} V_c & \text{for } x_{min} \le x \le x_{max} \text{ and } y_{min} \le y \le y_{max} \\ 0 & \text{otherwise} \end{cases} \tag{19}$$

$$\text{Let } j' = (n_y - 1)/4 + 1 \text{ then}$$

$$V(x, y) = \begin{cases} V_{i,j'} = V_{i,j'+1} = 0 \text{ for } (x_1 \le x_i \text{ and } x_i \le x_2) \text{ or } (x_3 \le x_i \text{ and } x_i \le x_4) \\ V_{i,j'} = V_{i,j'+1} = V_c \text{ otherwise} \\ V_{i,j} = 0 \text{ for } j \ne (j' \text{ or } j'+1) \end{cases} \tag{20}$$

Specifically for the double slit case, V is only non-zero for y-locations given by $j = j'$ or $j = j' + 1$ and for x-positions not coincident with one of the slits. This thus simulates a thin two mesh points wide plate at a fixed y-position, with adjustable slit openings, which span (x_1, x_2) and (x_3, x_4).

5.3 Initial Conditions and Potentials Implementation

The input parameters idtype and vtype are integers that select which initial data type and potential type, respectively, are to be used. Dependent on the type, elements of the associated parameter vector will be used to define the initial data or potential. Specifically, implementation options as follows:

Initial Data Types

1. Exact Family (16): idtype == 0
   - idpar(1) = m_x
   - idpar(2) = m_y
2. Boosted Gaussian (17): idtype == 1
   - idpar(1) = x_0
   - idpar(2) = y_0
   - idpar(3) = δ_x
   - idpar(4) = δ_y
   - idpar(5) = p_x
   - idpar(6) = p_y

Potential Types

1. No Potential (18): vtype == 0
2. Rectangular Barrier or Well (19): vtype == 1

- idpar(1) = x_{min}
- idpar(2) = x_{max}
- idpar(3) = y_{min}
- idpar(4) = y_{max}
- idpar(5) = V_c

3. Double Split (20): vtype == 2

- idpar(1) = x_1
- idpar(2) = x_2
- idpar(3) = x_3
- idpar(4) = x_4
- idpar(5) = V_c

The initial conditions and potentials that were previously discussed were implemented with the following code inside the sch_2d_adim script.

```
% Initial Data
if idtype == 0
    for idx = 1:nx
        psi(1, idx, :) = sin(idpar(1) * pi * x(idx)) * sin(idpar(2) * pi * y);
    end
elseif idtype == 1
    for idx = 1:nx
        psi(1, idx, :) = exp(1i*idpar(5)*x(idx)) * ...
            exp(1i*idpar(6)*y) .* ...
            exp(-((x(idx) - idpar(1))^2 / idpar(3)^2 +...
            (y - idpar(2)).^2 / idpar(4)^2));
    end
else
    fprintf('Invalid idtype');
    return
end

% Potential Data
if vtype == 0
    v = zeros(nx, ny);
elseif vtype == 1
    v = zeros(nx, ny);
    for ii = 1:nx
        for jj = 1:ny
            if vpar(1) <= x(ii) && x(ii) <= vpar(2) && ...
                    vpar(3) <= y(jj) && y(jj) <= vpar(4)
                v(ii, jj) = vpar(5);
            end
```

```
        end
    end
elseif vtype == 2
    v = zeros(nx, ny);
    jp = (ny-1)/4 + 1;
    v(:, jp) = vpar(5);
    v(:, jp+1) = vpar(5);
    for idx = 1:nx
        if (vpar(1) <= x(idx) && x(idx) <= vpar(2)) || ...
                (vpar(3) <= x(idx) && x(idx) <= vpar(4))
            v(idx, jp) = 0;
            v(idx, jp+1) = 0;
        end
    end
else
    fprintf('Invalid idtype');
    return
end
```

5.4 Alternating Direction Implicit Discretization

The ADI method is a finite difference method used for numerically solving partial differential equations using discretization. As for the 1d case, the continuum domain is discretized by introducing the discretization level, l, and the ratio of temporal to spatial mesh spacings

$$\lambda = \frac{\Delta t}{\Delta x} = \frac{\Delta t}{\Delta y}$$

We can then define the following parameters

$$n_x = n_y = 2^l + 1$$
$$\Delta x = \Delta y = 2^{-l}$$
$$\Delta t = \lambda \Delta x$$
$$n_t = \text{round}(t_{max}/\Delta t) + 1$$

We next define the difference operators ∂^h_{xx} and $\partial^h_{yy} by$

$$\partial^h_{xx} u^n_{i,j} = \frac{u^n_{i+1,j} - 2u^n_{i,j} + u^n_{i-1,j}}{\Delta x^2}$$
$$\partial^h_{yy} u^n_{i,j} = \frac{u^n_{i,j+1} - 2u^n_{i,j} + u^n_{i,j-1}}{\Delta y^2}$$

Using an ADI discretization of (14) we can get the following result. Note that the ADI method allows us to solve for one time step by using an intermediate half time step. This then results in two discretized equations that need to be solved. This can be done by solving n traditional systems.

$$\left(1 - i\frac{\Delta t}{2}\partial^h_{xx}\right)\psi^{n+1/2}_{i,j} = \left(1 + i\frac{\Delta t}{2}\partial^h_{xx}\right)\left(1 + i\frac{\Delta t}{2}\partial^h_{yy} - i\frac{\Delta t}{2}V_{i,j}\right)\psi^h_{i,j} \tag{21}$$

where

$$i = 2, 3, ..., n_x - 1, \quad j = 2, 3, ..., n_y - 1, \quad n = 1, 2, ..., n_t - 1$$

and

$$\left(1 - i\frac{\Delta t}{2}\partial^h_{yy} + i\frac{\Delta t}{2}V_{i,j}\right)\psi^{n+1}_{i,j} = \psi^{n+1/2}_{i,j} \tag{22}$$

where

$$i = 2, 3, ..., n_x - 1, \quad j = 2, 3, ..., n_y - 1, \quad n = 1, 2, ..., n_t - 1$$

Equations (21) and (22) are supplemented with the initial conditions

$$\psi^1_{i,j} = \psi_0(x_i, y_j)$$

and the boundary conditions

$$\psi^n_{1,j} = \psi^n_{n_x,j} = \psi^n_{i,1} = \psi^n_{i,n_y} = 0$$

The solution to these equation can be found using tri-diagonal systems. We rewrite (21) and (22) in the following forms respectively

$$c^+_i\psi^{n+1/2}_{i+1,j} + c^0_i\psi^{n+1/2}_{i,j} + c^-_i\psi^{n+1/2}_{i-1,j} = f_i$$
$$c^+_j\psi^{n+1}_{i,j+1} + c^0_j\psi^{n+1}_{i,j} + c^-_j\psi^{n+1}_{i,j-1} = f_j$$

where the different c vectors are the values along the three central diagonals in the sparse matrices. After rearranging (21) we find

$$c^+_i = \frac{-i\Delta t}{2\Delta x^2}$$
$$c^0_i = 1 + \frac{i\Delta t}{\Delta x^2}$$
$$c^-_i = \frac{-i\Delta t}{2\Delta x^2} = c^+_i$$
$$f_i = \left(1 + i\frac{\Delta t}{2}\partial^h_{xx}\right)\left(1 + i\frac{\Delta t}{2}\partial^h_{yy} - i\frac{\Delta t}{2}V_{i,j}\right)\psi^h_{i,j}$$

We can use these coefficient vectors to generate a sparse matrix using "spdiags" and then solve the system using left division for the first half step. Note that because $n_x = n_y$ some of the equations are the same. After rearranging (22) we find

$$c^+_j = c^+_i$$
$$c^0_j = 1 + \frac{i\Delta t}{\Delta y^2} + i\frac{\Delta t}{2}V_{i,j} = c^0_i + i\frac{\Delta t}{2}V_{i,j}$$

$$c_j^- = c_j^+$$
$$f_j = \psi_{i,j}^{n+1/2}$$

We can use these coefficient vectors to generate a sparse matrix using "spdiags" and then solve the system using left division for the first full step using the result from the half step.

5.5 Alternating Direction Implicit Discretization Implementation

The steps for the ADI discretization method previously discussed was implemented with the following code inside the sch_2d_adi.m script.

```
% Initialize storage for sparse matrix 1,2 and RHS 1,2
dl = zeros(nx,1);
d  = zeros(nx,1);
du = zeros(nx,1);
f1  = zeros(nx,1);
f2  = zeros(ny,1);

% Set up tridiagonal system 1
dl = -0.5i * dt / dx^2 * ones(nx, 1);
d  = (1 + 1i * dt / dx^2) .* ones(nx,1);
du = dl;

% Fix up boundary cases
d(1) = 1.0;
du(2) = 0.0;
dl(nx-1) = 0.0;
d(nx) = 1.0;

% Define sparse matrix 1
A_half = spdiags([dl d du], -1:1, nx, nx);

% Set up tridiagonal system 2 in the loop
% dl and du unchanged because ny=nx
% d is x and y dept. so add potential in the loop

% Compute solution using ADI scheme
for n = 1 : nt-1
    % Solve half step system
    psi_half = zeros(nx, ny);
    for k = 2:ny-1
        % Define RHS of linear system
        dyy = (psi(n,:,k+1) - 2*psi(n,:,k) + psi(n,:,k-1)) / dy^2;
        % right bracket calculation (intermediate value)
        psi_inter = psi(n,:,k) + 0.5i*dt*dyy - 0.5i*dt*v(:,k).'.*psi(n,:,k);

        % full RHS calculation
        dxx = (psi_inter(1:nx-2) - 2*psi_inter(2:nx-1) + psi_inter(3:nx)) / dx^2;
        f1(2:nx-1) = psi_inter(2:nx-1) + 0.5i*dt*dxx;

        % BCs
        f1(1) = 0.0;
```

```
            f1(ny) = 0.0;

            % Solve system 1, thus updating approximation to next time step
            psi_half(:, k) = A_half \ f1;
        end

        % Solve one step system
        for h = 2:nx-1

            % add potential to sparse matrix
            dv = d + 0.5i*dt*v(h,:).';
            % Fix up boundary cases
            dv(1) = 1.0;
            dv(nx) = 1.0;
            % Define sparse matrix 2 with potential
            A_one = spdiags([dl dv du], -1:1, ny, ny);

            % Define RHS of linear system
            f2 = psi_half(h, :).';

            % BCs
            f2(1) = 0.0;
            f2(nx) = 0.0;

            % Solve system 2, thus updating approximation to next time step
            psi(n+1, h, :) = A_one \ f2;
        end
    end
```

5.6 Convergence Testing

We then performed convergence tests in the same manner that was done for the one-dimensional case, taking into account the extra dimension. In particular, when performing a convergence test, compute the two-level deviation norms

$$\|d\psi^l\|_2(t^n) \tag{23}$$

as well as the deviations from the exact solution when using initial data corresponding to an exact solution

$$\|E(\psi^l)\|_2(t^n) = \|\psi_{exact} - \psi^l\|_2(t^n)$$

Note that for the two-dimensional case, the spatial norm $\|.\|_2$ involves a sum over both spatial dimensions, i.e. treat any 2d grid function as a vector of length $n_x \times n_y$.

5.7 Convergence Testing Implementation

The convergence test we preformed had the following parameters

1. idtype = 0, vtype = 0

- idpar = [2, 3]

- tmax = 0.05
- lambda = 0.05
- $l_{min} = 6$
- $l_{max} = 9$

The following is the function that will calculate the convergence test and plot the three graphs. The code was derived from the previous section.

```
function ctest_2d()
    % get solutions for convtest for different levels
    mx = 2;
    my = 3;
    [x6 y6 t6 psi6 psire6 psiim6 psimod6 v6] = ...
        sch_2d_adi(0.05, 6, 0.05, 0, [mx, my], 0, []);
    [x7 y7 t7 psi7 psire7 psiim7 psimod7 v7] = ...
        sch_2d_adi(0.05, 7, 0.05, 0, [mx, my], 0, []);
    [x8 y8 t8 psi8 psire8 psiim8 psimod8 v8] = ...
        sch_2d_adi(0.05, 8, 0.05, 0, [mx, my], 0, []);
    [x9 y9 t9 psi9 psire9 psiim9 psimod9 v9] = ...
        sch_2d_adi(0.05, 9, 0.05, 0, [mx, my], 0, []);

    % coarsen solutions to match size of lmin solution
    psi7 = psi7(1:2:end, 1:2:end, 1:2:end);
    psi8 = psi8(1:4:end, 1:4:end, 1:4:end);
    psi9 = psi9(1:8:end, 1:8:end, 1:8:end);

    dpsi6 = psi7 - psi6;
    dpsi7 = psi8 - psi7;
    dpsi8 = psi9 - psi8;

    % calculate rms values of dpsi
    norm_dpsi6 = psi_norm(dpsi6);
    norm_dpsi7 = psi_norm(dpsi7);
    norm_dpsi8 = psi_norm(dpsi8);

    % Removed all non-plot lines for clarity
    plot(t6, norm_dpsi6, 'r-.o');
    plot(t6, 4 * norm_dpsi7, 'g-.+');
    plot(t6, 4^2 * norm_dpsi8, 'b-.*');

    % calculate exact solution
    psi_exact = zeros(size(t6, 2), size(x6, 2), size(y6, 2));
    for idt = 1:size(t6, 2)
        for idx = 1:size(x6, 2)
            psi_exact(idt, idx, :) = exp(-1i*(mx^2+my^2)*pi^2*t6(idt)) * ...
                sin(mx*pi*x6(idx)) * sin(my*pi*y6);
        end
    end

    % calculate rms values of E
    norm_E6 = psi_norm(psi_exact - psi6);
```

```
    norm_E7 = psi_norm(psi_exact - psi7);
    norm_E8 = psi_norm(psi_exact - psi8);
    norm_E9 = psi_norm(psi_exact - psi9);

    % Removed all non-plot lines for clarity
    plot(t6, norm_E6, 'r-.o');
    plot(t6, 4 * norm_E7, 'g-.+');
    plot(t6, 4^2 * norm_E8, 'b-.*');
    plot(t6, 4^3 * norm_E9, 'c-.*');

end

% Output
% norm: Vector of norms [nt]
function norm = psi_norm(psi)
    [nt, nx, ny] = size(psi);
    norm = zeros(nt, 1);
    for n = 1:nt
        for ii = 1:nx
            for jj = 1:ny
                norm(n) = norm(n) + abs(psi(n, ii, jj))^2;
            end
        end
    end
    norm = sqrt(norm / nx / ny);
end
```

5.8 Numerical Experiments

The goal for the numerical experiments was to make AVI movies of the following scenarios. In all cases we used Gaussian initial data, with or without a boost (in either and/or both directions). Two types of plot will be made, first a contour plot and second was a surface plot. Note that a level of 8 was user for all experiments.

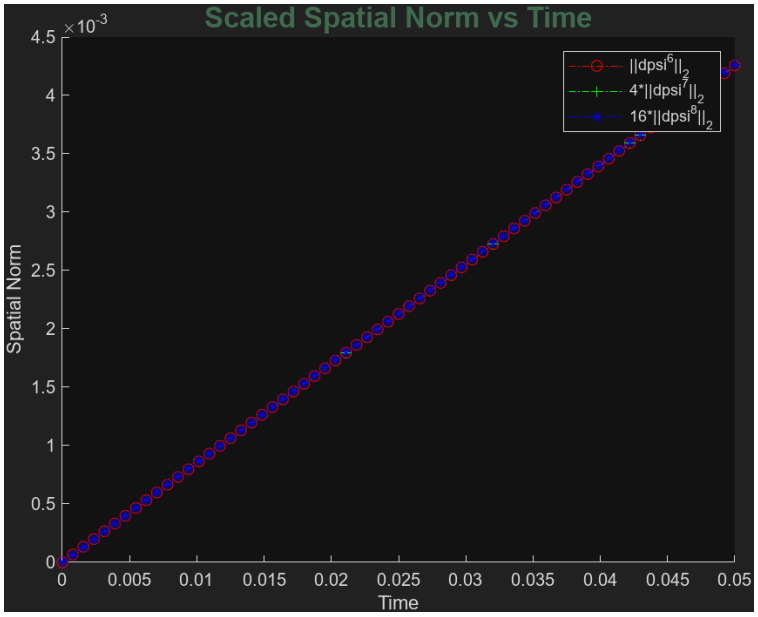
1. Scattering off a rectangular barrier (idtype = 1, vtype = 1).
   - tmax = 0.04, lambda = 0.01
   - $x_0 = 0.3, y_0 = 0.5, \delta_x = 0.055, \delta_y = 0.055, p_x = 20, p_y = 10$
   - $x_{min} = 0.5, x_{max} = 0.7, y_{min} = 0.2, y_{max} = 0.8, V_c = 10^6$
2. Scattering off a rectangular well (idtype = 1, vtype = 1).
   - tmax = 0.04, lambda = 0.01
   - $x_0 = 0.3, y_0 = 0.5, \delta_x = 0.055, \delta_y = 0.055, p_x = 0, p_y = 0$
   - $x_{min} = 0.5, x_{max} = 0.7, y_{min} = 0.2, y_{max} = 0.8, V_c = -10^4$
3. Scattering through a double slit (idtype = 1, vtype = 2).
   - tmax = 0.04, lambda = 0.01
   - $x_0 = 0.5, y_0 = 0.1, \delta_x = 0.055, \delta_y = 0.055, p_x = 0, p_y = 15$
   - $x_1 = 0.3, x_2 = 0.4, x_3 = 0.6, x_4 = 0.7, V_c = 10^6$

The code for all the plotting can be seen in movies.m. For the barrier experiment we can see the wave come in from the right side and scatter off the barrier and no part of the wave makes it through. This makes sense due to it having such a large potential in that rectangular section. Looking at the contour plot we can confirm that zero values can be seen inside the barrier rectangle. For the well experiment we can see the wave come in from the right side and scatter off the barrier and some part of the wave makes it through. This makes sense due to it having such a large negative potential in that rectangular section. Note that some of the wave function was able to make it through and exist which makes sense due to the well nature. Looking at the contour plot we can confirm that small local extremes can be seen inside the well rectangle. For the double slit experiment we can see the wave come in from the left side and scatter off the barrier and a small part of the wave makes it through the two slits and then interferes with itself. We can then see the bright and dark fringes at the far end of the plot which is a characteristic of the double slit experiment. Looking at the contour plot we can confirm that the wave function leaks through the double slits. The surface and contour plots can be found in the submission folder.

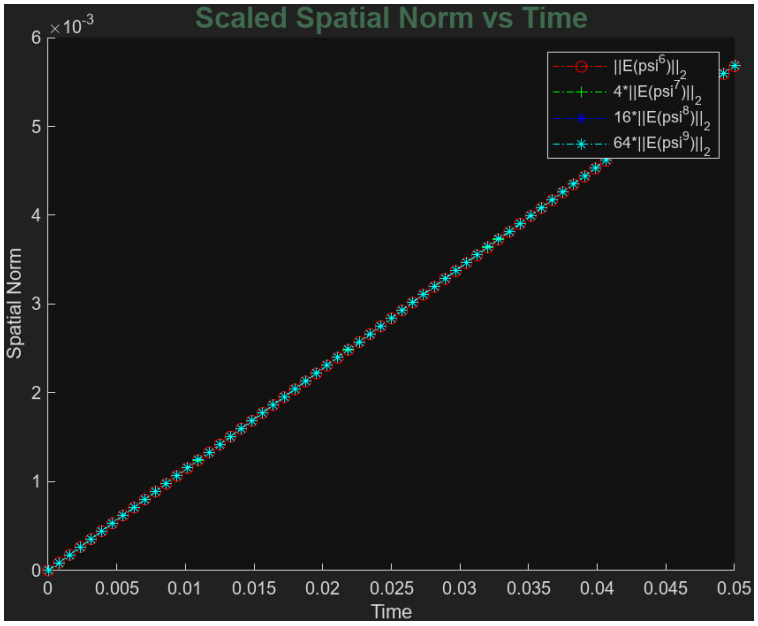
6 Problem 2 Results

6.1 Convergence Test

The result of the 4-level convergence test can be seen blow. Note that we do indeed see near-coincidence of the curves, with better agreement as we go to higher values of l. As a stringent test that our numerical solution is behaving as it should, implying that the implementation is correct, we performed a 4-level (6, 7, 8 and 9) convergence test for our FDA as discussed in class following the steps discussed in the previous section, the figures can be seen below.



The following graph is for the case of using ψ_{exact} in our 4-level convergence test calculation.



If we wanted to complete a more detailed convergence test we could extend our convergence test by repeating the calculation with an even higher levels.

6.2 Numerical Experiments

The results for the numerical experiments can be seen at their respective links or in the movies folder of the submission. Two types of plot were be made, first a contour plot and second a surface plot for each of the three cases. The following are snap shots of each wave function for the surface and contour plots. For the barrier movie we can see the wave come in from the right side and scatter off the barrier and no part of the wave makes it through. This makes sense due to it having such a large potential in that rectangular section. For the well movie we can see the wave come in from the right side and scatter off the barrier and some part of the wave makes it through. This makes sense due to it having such a large negative potential in that rectangular section. Note that some of the wave function was able to make it though and exist in which makes sense due to the well nature. For the double slit movie we can see the wave come in from the left side and scatter off the barrier and a small part of the wave makes it through the two slits and then interferes with itself. We can then see the bright and dark fringes at the far end of the plot which is a characteristic of the double slit experiment.

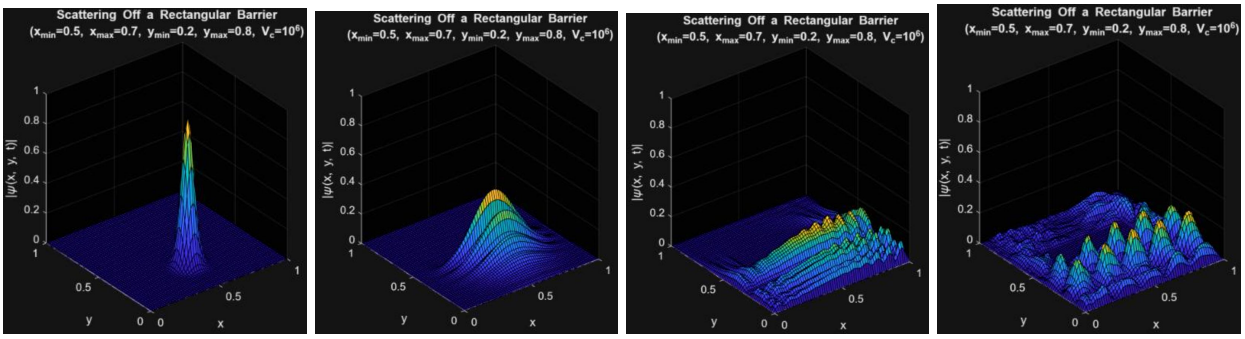


Figure 1: Scattering off a rectangular barrier

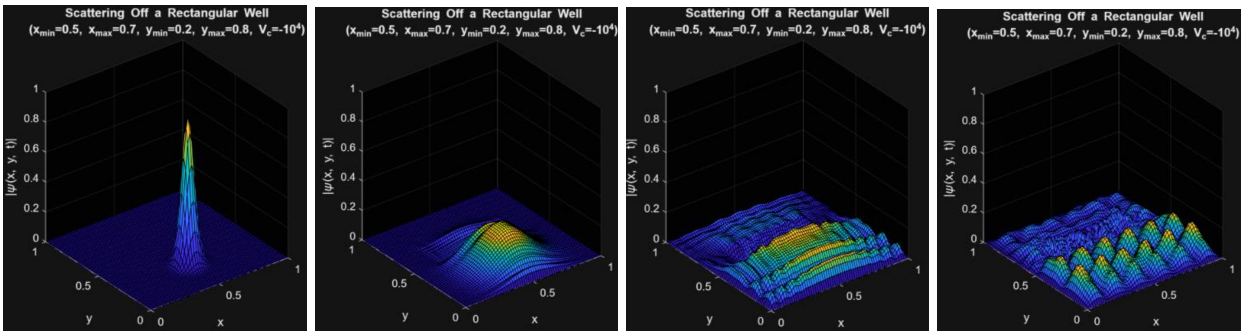


Figure 2: Scattering off a rectangular well

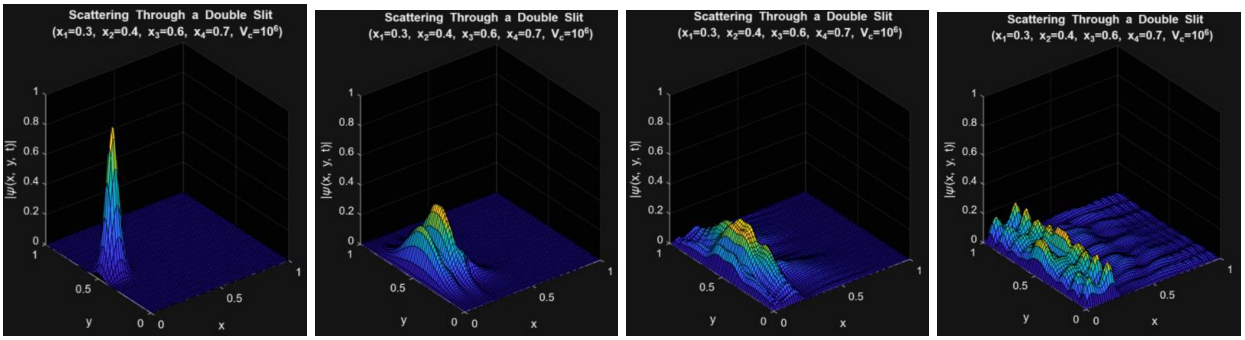


Figure 3: Scattering through a double slit

The following are snap shots of each wave function for the contour plots.

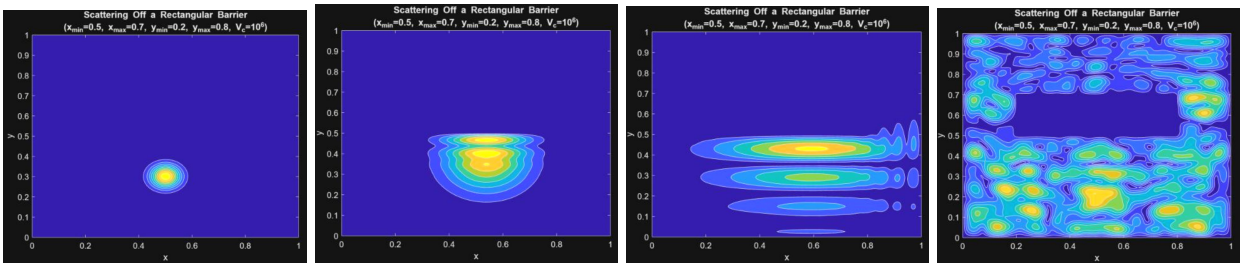


Figure 4: Scattering off a rectangular barrier

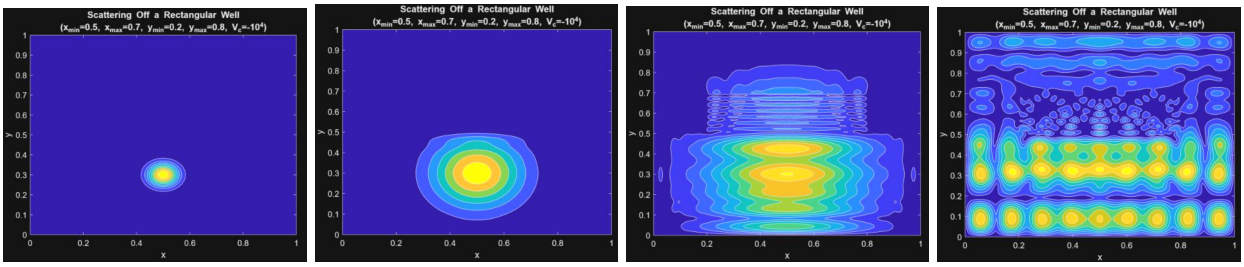


Figure 5: Scattering off a rectangular well

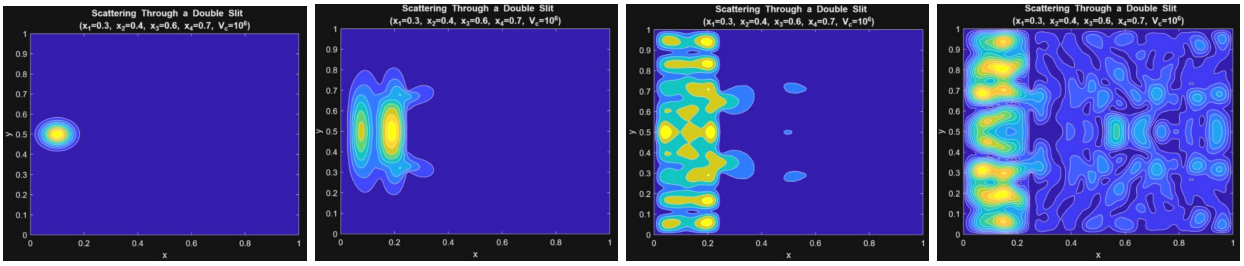


Figure 6: Scattering through a double slit

7 Discussion & Conclusion

In this project we analyzed two problems, the first being solving the one-dimensional time-dependent Schrodinger equation using the Crank-Nicolson discretization method and the second being solving the two-dimensional Schrodinger equation using the alternating direction implicit (ADI) discretization technique.

First, we developed an algorithm to use the Crank-Nicolson discretization method which includes solving a tri-diagonal matrix. We solved for the coefficients based off the one-dimensional time-dependent Schrodinger equation and then used "spdiags" to generate the matrix. We then solved the system using left division and calculated some useful quantities like the running integral of the probability density. With the Crank-Nicolson algorithm developed, a 4-level convergence test was performed where near-coincidence of the curves was observed, with better agreement as we go to higher values of l. This was done again but comparing the exact value of the solution and again near-coincidence of the curves was observed. We then wanted on to perform two numerical experiments where the excess fractional probability that the particle spends in a given spatial interval was analyzed. For the experiments described, this quantity will span orders of magnitude so, particularly for the purposes of plotting, it was convenient to compute its natural logarithm. These values were graphed for the two experiments. It made sense that for the barrier survey, the graph started a bit below zero and then dropped off with increasing values of V_0. For the well survey, it made sense that it started a bit above zero and then exhibited some sinusoidal tendencies while also dropping in value as we increased V_0.

We then moved into implementing an algorithm to use the ADI method which includes solving a multiple tri-diagonal matrices. This method uses a half step in n in order to be able to calculate a full step in n. This is why multiple tri-diagonal matrices were needed to be constructed. We solved for the coefficients based off the two-dimensional time-dependent Schrodinger equation which turned into two equations when discretized. We then used "spdiags" to generate the matrices. Next, we solved the systems using left division and calculated some useful quantities like the real and imaginary part of the wave function. As done in the first problem, a 4-level convergence test was performed where near-coincidence of the curves was observed, with better agreement as we go to higher values of l. This was done again but comparing the exact value of the solution and again near-coincidence of the curves was observed. Lastly, we performed three numerical experiments where we analyzed scattering off a rectangular barrier, scattering off a rectangular well, and scattering through a double slit. For each of these experiments a surface and contour plot was generated. For the barrier movie we can see the wave come in from the right side and scatter off the barrier and no part of the wave makes it through. This makes sense due to it having such a large potential in that rectangular section. For the well movie we can see the wave come in from the right side and scatter off the barrier and some part of the wave makes it through. This makes sense due to it having such a large negative potential in that rectangular section. Note that some of the wave function was able to make it through and exist which makes sense due to the well nature. For the double slit movie we can see the wave come in from the left side and scatter off the barrier and a small part of the wave makes it through the two slits and then interferes with itself. We can then see the bright and dark fringes at the far end of the plot which is a characteristic of the double slit experiment.

I did not use generative AI. I encountered a couple problems during the implementation of this project. The first problem I encountered was not knowing that "i" behaves weirdly when you use it as an indexing variable and representing the square root of -1. This was easily solved by not using "i" to index the loops. A second problem I encountered was generating the 251 uniformly spaced values of $\ln(V_0)$ incorrectly for the barrier survey. I generated them with the following command which does not space the values correctly:

```
V0 = linspace(exp(-2), exp(5), num_exp);
```

when I should have generated the values like this:

```
lnV0 = linspace(-2, 5, num_exp);
V0 = exp(lnV0);
```

The last problem I had was getting proper convergence in the 2-d case but that was because I calculated one of the coefficients wrong from my initial derivation. This was easily fixed once the type was found.